Estruturas de Dados – Aula 09

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Árvores

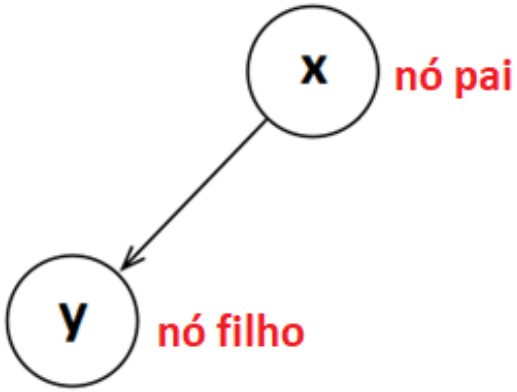
- Frequentemente, uma árvore é considerada uma extensão do conceito de lista, ou seja, uma estrutura de árvore pode ser pensada como uma generalização de uma lista.

- Assim como uma lista, uma árvore é uma estrutura de dados recursiva.

Árvores

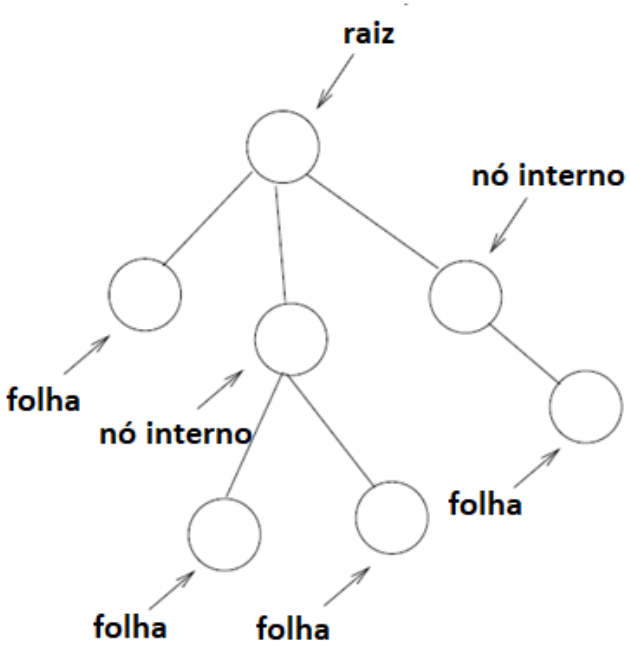
- Árvores são estruturas de dados que caracterizam uma relação hierárquica (não linear) entre seus elementos, chamados nós.

- Essa hierarquia é tal que se um nó x é imediatamente superior a um nó y, então dizemos que x é pai de y e y é filho de x.



Árvores

- O único nó que não tem pai é chamado de raiz da árvore, e todo nó que não tem filho é chamado de folha (ou nó terminal).



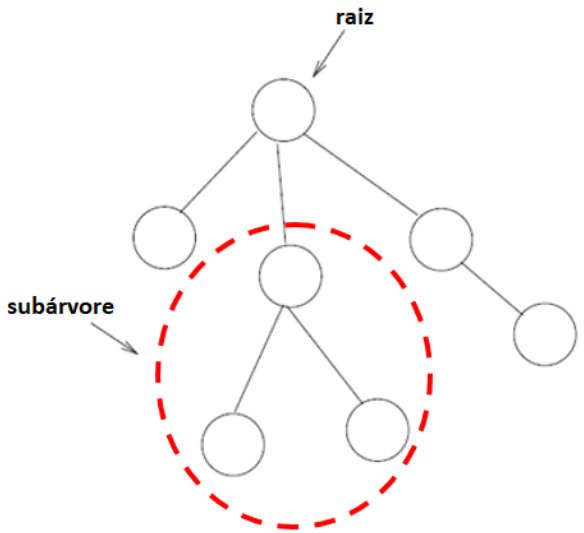
- O nó que não é raiz e não é folha é chamado nó interno (ou nó não terminal).

Árvores

Definição recursiva

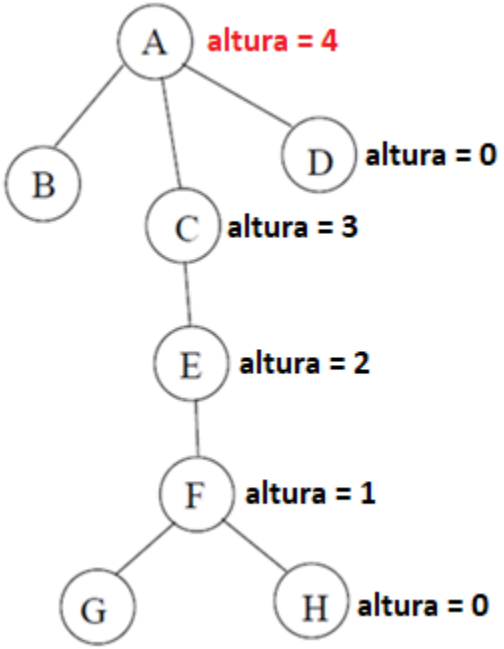
Uma árvore do tipo T é constituída por:

- uma árvore vazia, ou;
- um nó do tipo T, chamado de raiz, com um número finito de árvores do tipo T associadas, chamadas subárvores.



Altura da árvore

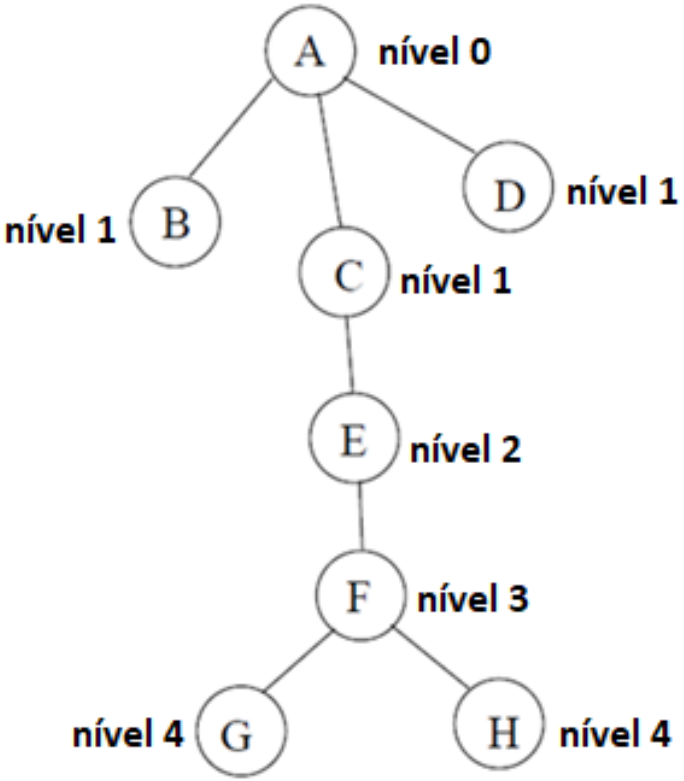
- A altura de um nó *x* da árvore é o número de arestas do caminho mais longo, a partir do nó *x*, até uma folha.



- A altura da árvore é a altura do nó raiz da respectiva árvore.

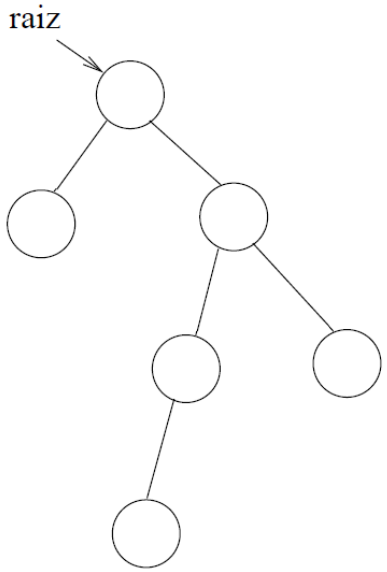
Nível do nó

- O nível de um nó *x* da árvore é o número de arestas até a raiz da árvore.



Árvores binárias

- Uma árvore é binária se cada um de seus nós tem no máximo 2 filhos.



- Os filhos (ou subárvores) de um nó em uma árvore são chamados de filho esquerdo e filho direito.

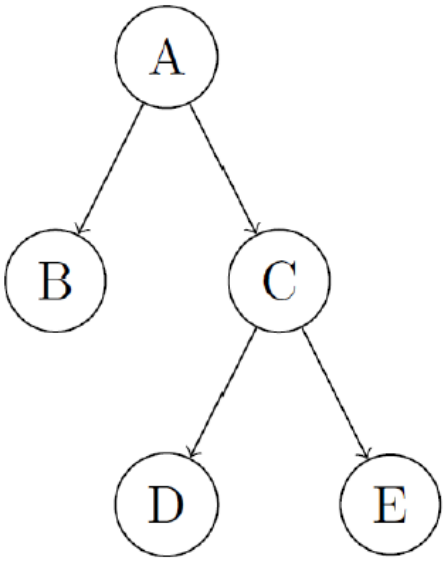
Implementação da árvore binária

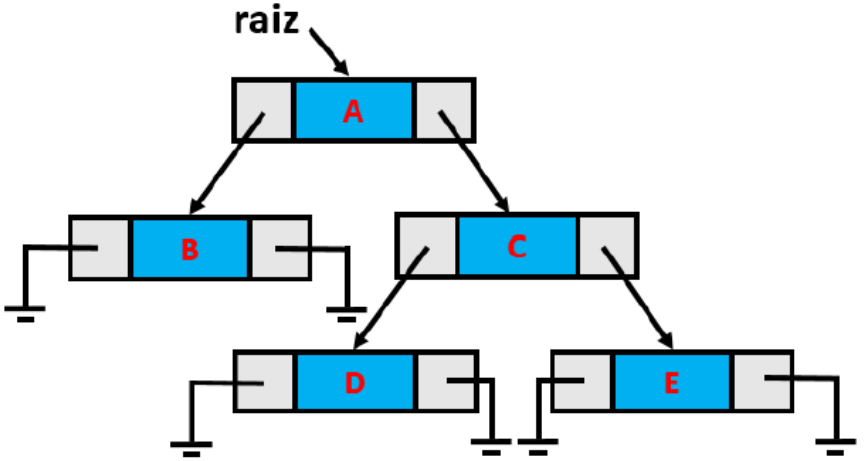
- Assim como as listas, uma árvore binária pode ser implementada de forma estática ou dinâmica.

- Mas na maioria das vezes optamos por trabalhar com a implementação dinâmica.

Implementação da árvore binária

Implementação dinâmica:



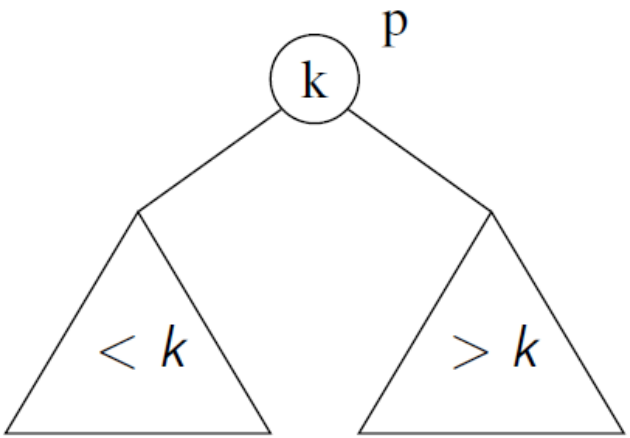


O endereço de uma árvore binária é o endereço do nó raiz da árvore.

Árvore binária de busca (ABB)

Uma árvore binária é chamada árvore binária de busca se para cada nó p vale que:

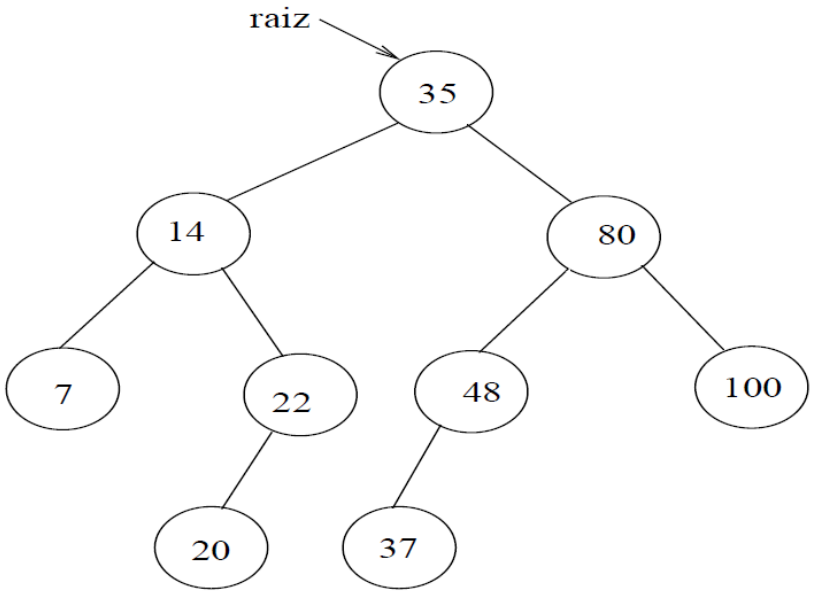
- todo nó da subárvore esquerda de p tem chave menor que a chave de p, e;
- todo nó da subárvore direita de p tem chave maior que a chave de p.



Árvore binária de busca (ABB)

- Em uma árvore binária de busca para qualquer nó p, se os filhos da esquerda e direita são não nulos, temos que:

$$p.esq.chave < p.chave < p.dir.chave$$

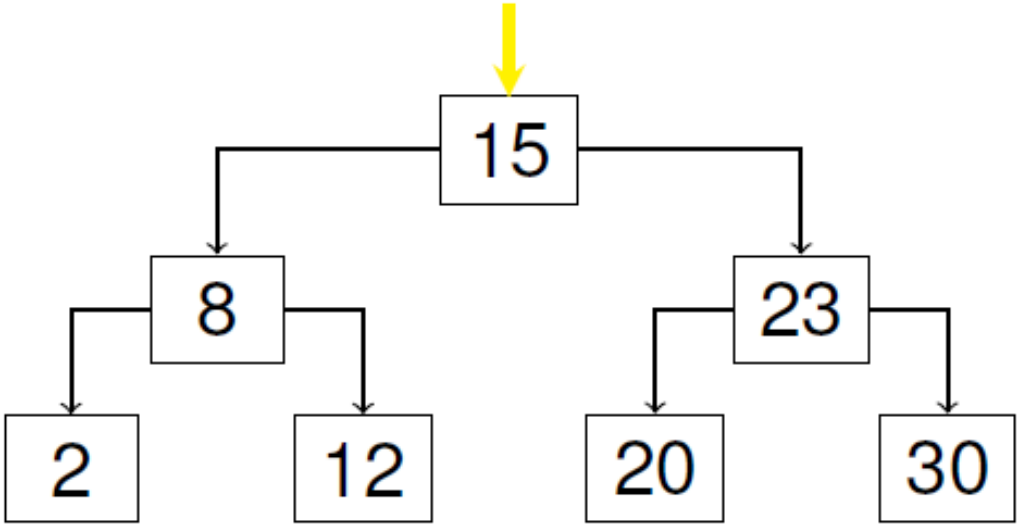


Principais operações em ABB

- **Inserção**
  - inserir um novo elemento na árvore binária de busca
- **Busca**
  - buscar/procurar um elemento na árvore binária de busca
- **Remoção**
  - excluir (se existir) um elemento na árvore binária de busca

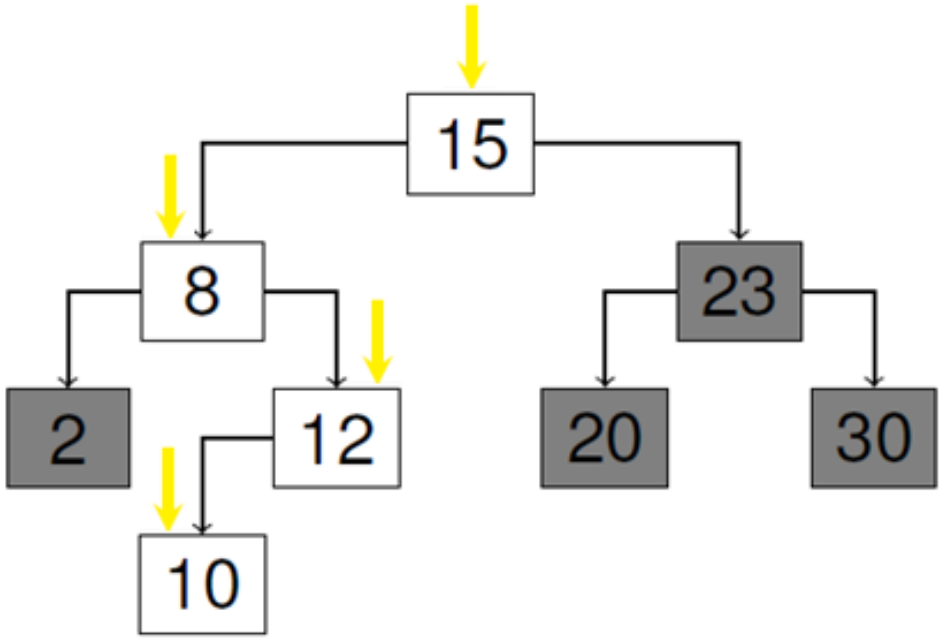
Inserção de um elemento na ABB

Vamos considerar a árvore binária de busca a seguir e inserir o valor 10. Perceba que devemos iniciar a inserção a partir da raiz.



Inserção de um elemento na ABB

Note que depois de inserir o valor 10 a árvore continua sendo binária de busca.



Inserção de um elemento na ABB

A função inserir recebe uma árvore binária de busca e um novo valor. Então a função cria um novo nó com o novo valor e insere na árvore de modo que a árvore resultante seja de busca, e por fim, retorna um ponteiro para a nova árvore.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct node* link;
5
6  struct node{ int chave; link esq; link dir; };
7
```

Inserção de um elemento na ABB

```
 8 link inserir(link raiz, int valor) {
 9     if (raiz == NULL) { //se a árvore estiver vazia
10         link novoNode = (link)malloc(sizeof(struct node)); //então cria um novo nó
11         novoNode->chave = valor;
12         novoNode->esq = NULL;
13         novoNode->dir = NULL;
14         return novoNode;
15     }
16     if (valor < raiz->chave) //se o valor for menor que a chave
17         raiz->esq = inserir(raiz->esq, valor); //insere na subárvore esquerda
18     else if (valor > raiz->chave) //se o valor for maior que a chave
19         raiz->dir = inserir(raiz->dir, valor); //insere na subárvore direita
20     return raiz; //retorna a raiz atualizada
21 }
22
```

Complexidade no pior caso: O(altura) = O(n)

Busca de um elemento na ABB

A função buscar recebe o endereço de uma árvore binária de busca e um valor chave, e retorna um ponteiro para o nó com a chave, se tal nó existir, ou NULL, caso contrário.

Busca de um elemento na ABB

```
23  link buscar(link raiz, int chave) {
24      //se a árvore estiver vazia ou encontrou a chave
25      if (raiz == NULL || raiz->chave == chave)
26          return raiz;
27      if (chave < raiz->chave) //se a chave for menor que a chave na raiz
28          return buscar(raiz->esq, chave); //busca na subárvore esquerda
29      else  //se a chave for maior que a chave na raiz
30          return buscar(raiz->dir, chave); //busca na subárvore direita
31  }
32
```

Complexidade no pior caso: O(altura) = O(n)

Remoção de um elemento na ABB

Para remover um elemento X em uma árvore binária de busca devemos considerar 3 situações:

(1) A remoção ocorre em um nó folha (sem filhos);

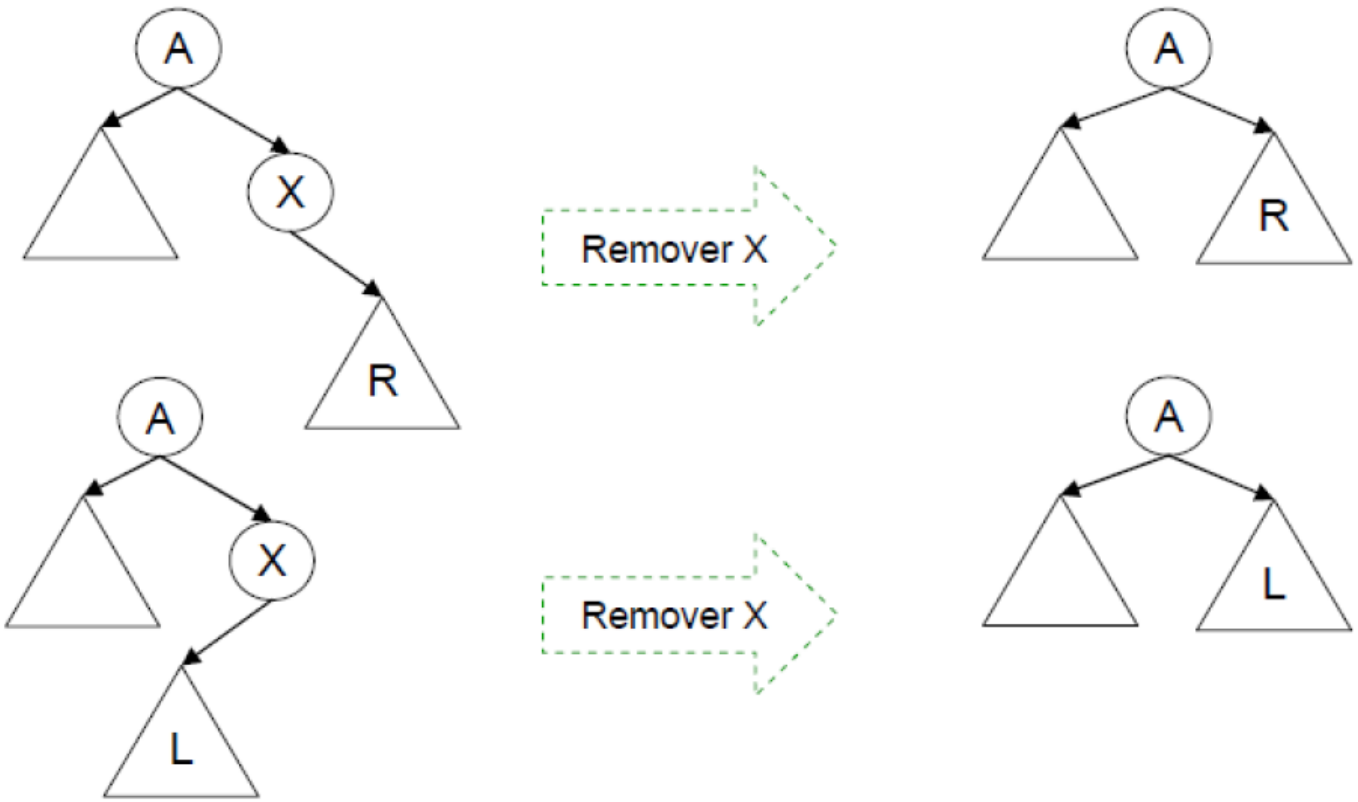
(2) A remoção ocorre em um nó com apenas 1 filho (1 subárvore);

(3) A remoção ocorre em um nó que tem 2 filhos (2 subárvores).

Temos que a 1ª situação é trivial. Mas precisamos entender melhor as outras duas situações.

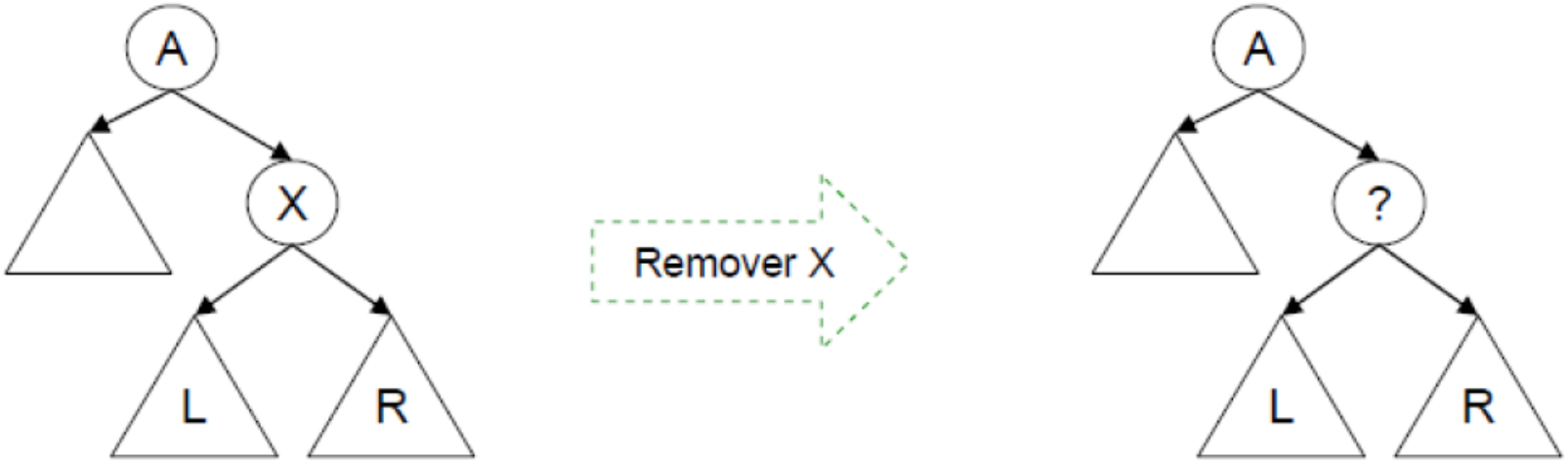
Remoção de um elemento na ABB

2ª situação:



Remoção de um elemento na ABB

3ª situação:



Remoção de um elemento na ABB

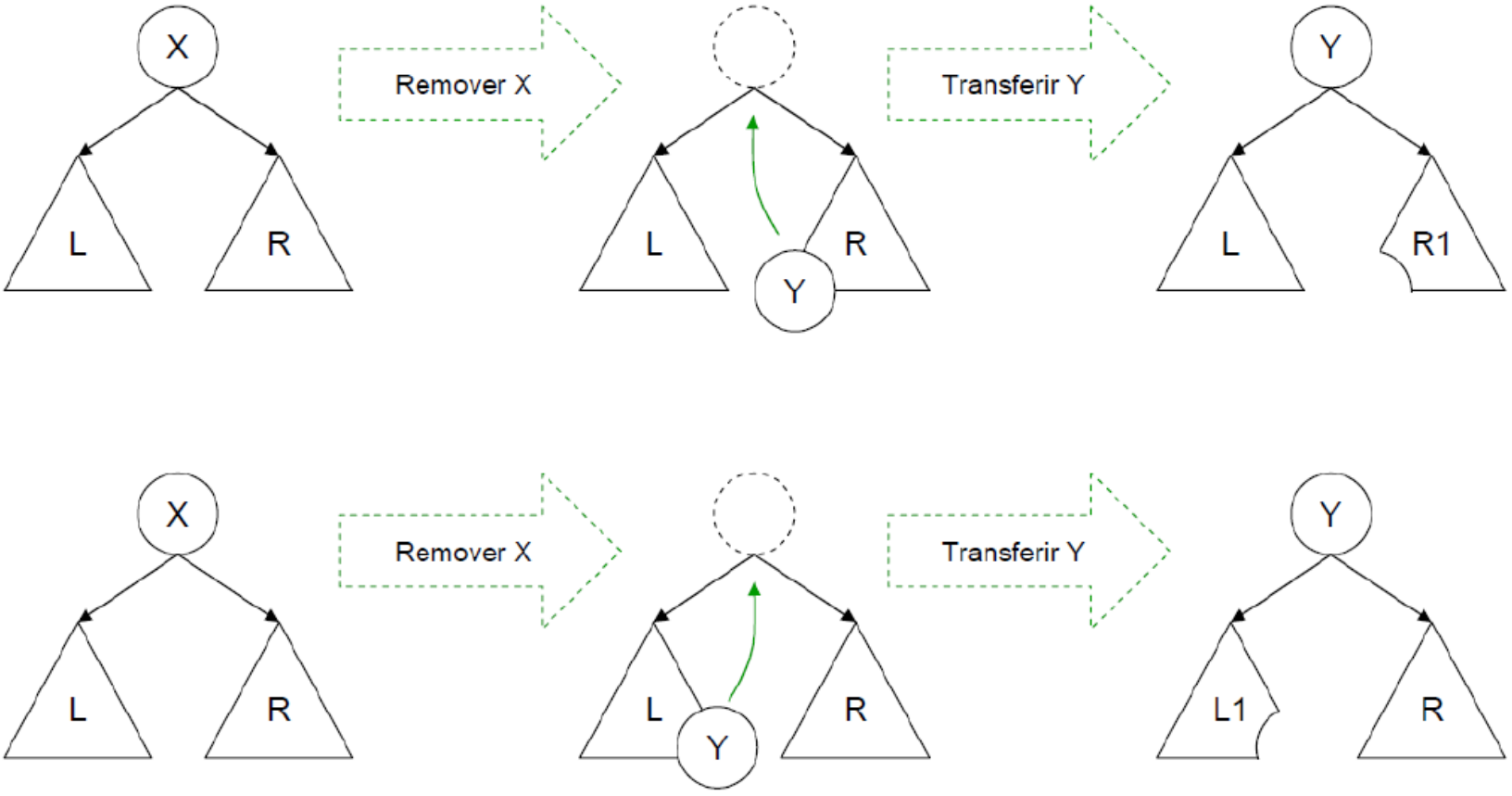
3ª situação:

Quando a remoção é um nó com 2 filhos (2 subárvores) existem 2 estratégias para remover o valor X preservando a propriedade da árvore binária de busca:

- Encontrar o elemento de menor valor Y (sucessor lógico) na subárvore direita de X, e transferi-lo para o nó ocupado por X;
- Encontrar o elemento de maior valor Y (predecessor lógico) na subárvore esquerda de X, e transferi-lo para o nó ocupado por X.

Remoção de um elemento na ABB

3ª situação:



Remoção de um elemento na ABB

Para implementarmos a função remover vamos considerar duas funções auxiliares para tratar as 3 situações de remoção para um nó na árvore binária de busca.

Remoção de um elemento na ABB

```
33  link removerCaso1e2(link raiz) { //casos 1 e 2 (não tem filho ou tem 1 filho)
34      link q = raiz;
35      if (raiz->esq == NULL){
36          q = raiz->dir; //caso 1 (se dir == NULL) ou caso 2 (se dir != NULL)
37          free(raiz);
38          return q; //retorna a raiz atualizada
39      } else if (raiz->dir == NULL){
40          q = raiz->esq; //caso 2: esq != NULL e dir == NULL
41          free(raiz);
42          return q; //retorna a raiz atualizada
43      }
44      return q;
45  }
46
```

Remoção de um elemento na ABB

```
47  link removerCaso3(link raiz) { //caso 3 (tem 2 filhos)
48      link p = raiz;
49      link q = raiz->dir;
50      while (q->esq != NULL) { //encontrar sucessor lógico
51          p = q;
52          q = q->esq;
53      }
54      if (p != raiz) { //sucessor não é filho da direita da raiz
55          p->esq = q->dir;
56          q->dir = raiz->dir;
57      }
58      q->esq = raiz->esq;
59      free(raiz);
60      return q; //retorna a raiz atualizada
61  }
62
```

Remoção de um elemento na ABB

```
63  link remover(link raiz, int chave) {
64      if (raiz == NULL) return NULL; //não encontrou a chave
65      if (chave == raiz->chave) //encontrou a chave
66          if (raiz->esq == NULL || raiz->dir == NULL) //caso 1 ou caso 2
67              raiz = removerCaso1e2(raiz);
68          else //caso 3
69              raiz = removerCaso3(raiz);
70      else //ainda não encontrou a chave
71          if (chave < raiz->chave) //se a chave for menor que a chave na raiz
72              raiz->esq = remover(raiz->esq, chave); //remove na subárvore esquerda
73          else //se a chave for maior que a chave na raiz
74              raiz->dir = remover(raiz->dir, chave); //remove na subárvore direita
75      return raiz;
76  }
77
```

Complexidade no pior caso: O(altura) = O(n)

Percurso em árvores binárias

- Um percurso é uma visita sistemática a cada um dos nós de uma árvore binária.

- Temos 3 percursos em profundidade bem conhecidos:
  - Percurso pré-ordem;
  - Percurso in-ordem (ou simétrico);
  - Percurso pós-ordem.

- Temos 1 percurso por nível (ou em largura).

Percurso pré-ordem

Segue-se recursivamente as seguintes operações

(i) visita a raiz

(ii) percorrer a subárvore esquerda em pré-ordem

(iii) percorrer a subárvore direita em pré-ordem

Percurso in-ordem (ou simétrico)

Segue-se recursivamente as seguintes operações

(i) percorrer a subárvore esquerda em in-ordem

(ii) visita a raiz

(iii) percorrer a subárvore direita em in-ordem

Percurso pós-ordem

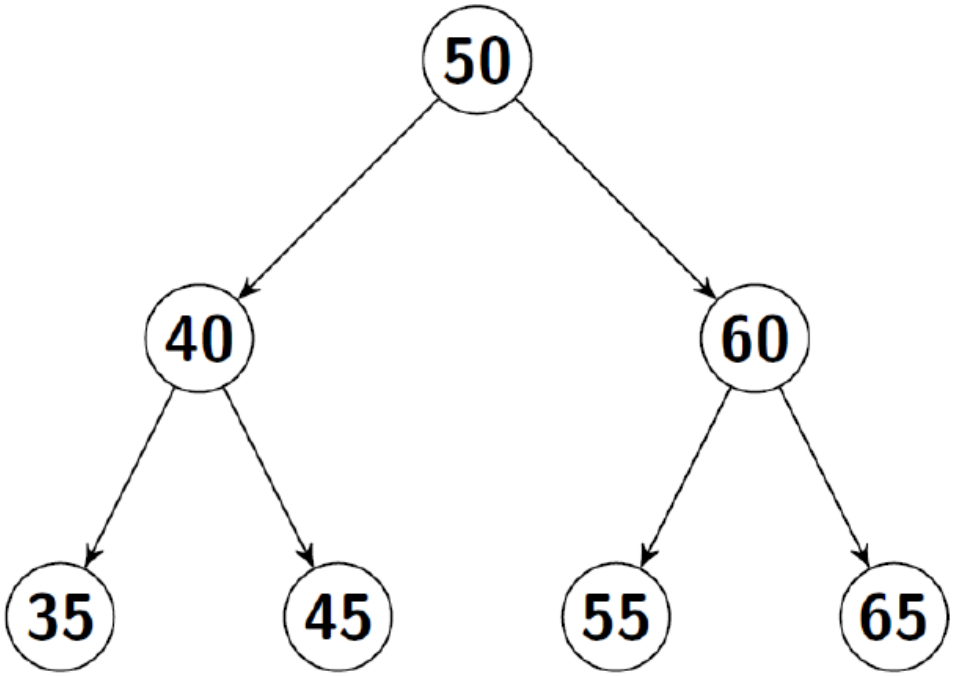
Segue-se recursivamente as seguintes operações

(i) percorrer a subárvore esquerda em pós-ordem

(ii) percorrer a subárvore direita em pós-ordem

(iii) visita a raiz

Percursos em profundidade



Pré-ordem: 50, 40, 35, 45, 60, 55, 65
In-ordem: 35, 40, 45, 50, 55, 60, 65 (**ordenado**)
Pós-ordem: 35, 45, 40, 55, 65, 60, 50

Implementação do percurso pré-ordem

```
78  void percursoPreOrdem(link raiz) {
79      if (raiz != NULL) {
80          printf("%i ", raiz->chave); //visita a raiz
81          percursoPreOrdem(raiz->esq); //percorre a subárvore esquerda
82          percursoPreOrdem(raiz->dir); //percorre a subárvore direita
83      }
84  }
85
```

Complexidade: $O(n)$

Implementação do percurso in-ordem

```
86  void percursoInOrdem(link raiz) {
87      if (raiz != NULL) {
88          percursoInOrdem(raiz->esq); //percorre a subárvore esquerda
89          printf("%i ", raiz->chave); //visita a raiz
90          percursoInOrdem(raiz->dir); //percorre a subárvore direita
91      }
92  }
93
```

Complexidade: $O(n)$

Implementação do percurso pós-ordem

```
94  void percursoPosOrdem(link raiz) {
95      if (raiz != NULL) {
96          percursoPosOrdem(raiz->esq); //percorre a subárvore esquerda
97          percursoPosOrdem(raiz->dir); //percorre a subárvore direita
98          printf("%i ", raiz->chave); //visita a raiz
99      }
100 }
101
```

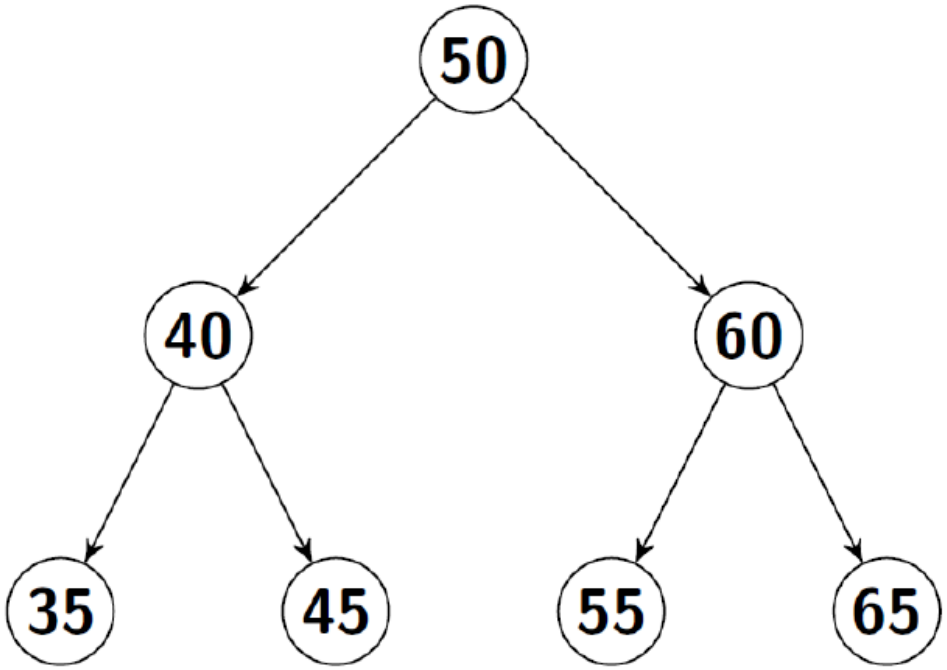
Complexidade: $O(n)$

Percurso por nível (ou em largura)

- Em um percurso por nível (ou em largura) os nós são visitados na ordem dos níveis da árvore, isto é, primeiro são visitados os nós do nível 0, depois do nível 1, depois do nível 2 e assim por diante.

- Devemos utilizar uma fila para implementar o percurso por nível (ou em largura).

Percurso por nível (ou em largura)



Largura (por Nível): 50, 40, 60, 35, 45, 55, 65

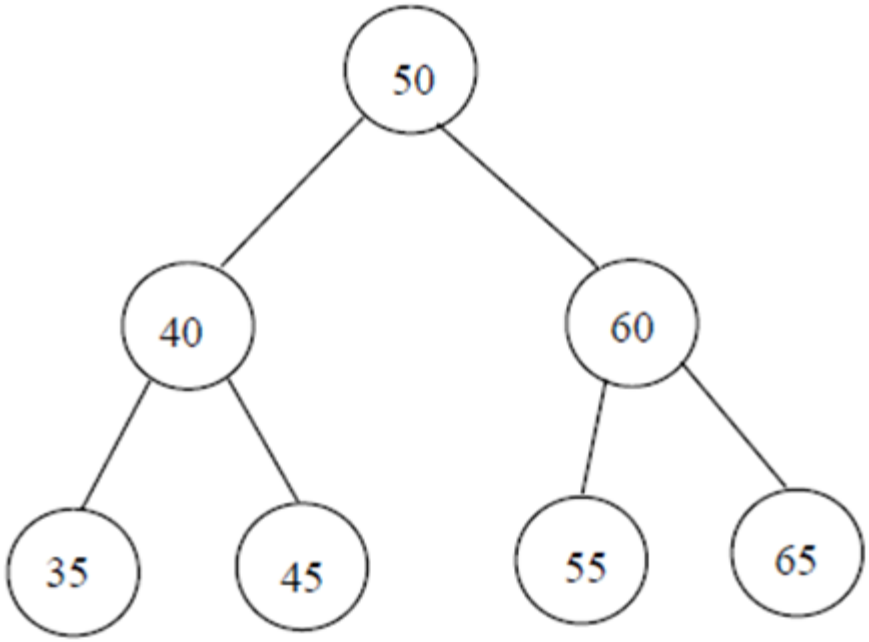
Impressão da árvore binária

Para imprimir uma árvore binária vamos considerar funções baseadas nos percursos pré-ordem e in-ordem.

Impressão da árvore binária

```
102 void imprimirPreOrdem(link raiz) {
103     printf("(");
104     if (raiz != NULL) {
105         printf("% i ", raiz->chave);
106         imprimirPreOrdem(raiz->esq);
107         imprimirPreOrdem(raiz->dir);
108     }
109     printf(")");
110 }
111
```

Impressão da árvore binária

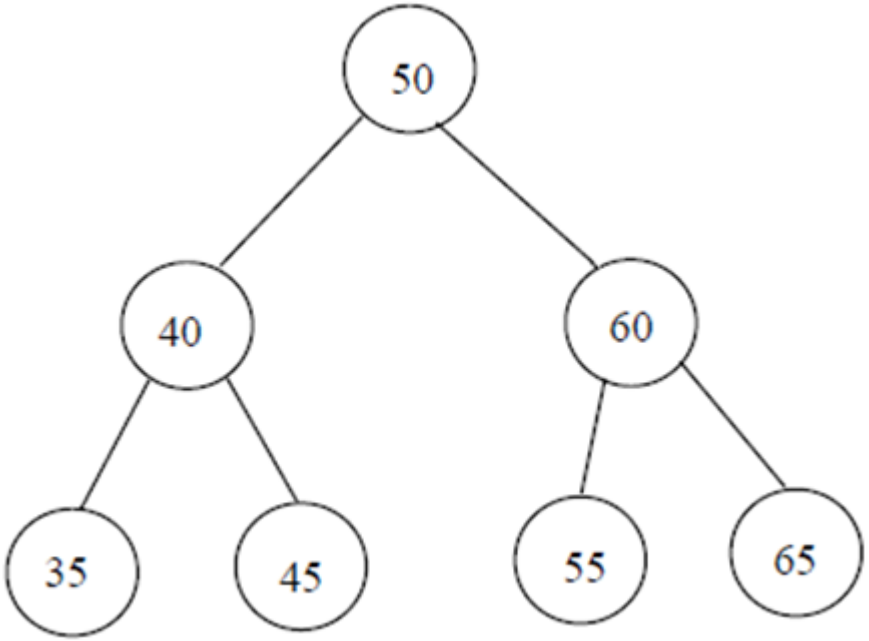



```
( 50 ( 40 ( 35 ()())( 45 ()()))( 60 ( 55 ()())( 65 ()())))
```

Impressão da árvore binária

```
112 void imprimirInOrdem(link p, int b) {
113     int i;
114     if (p == NULL) {
115         for (i = 0; i < b; i++) printf("-----");
116         printf("NULL\n");
117         return;
118     }
119     imprimirInOrdem(p->dir, b + 1);
120     for (i = 0; i < b; i++) printf("-----");
121     printf("%i\n", p->chave);
122     imprimirInOrdem(p->esq, b + 1);
123 }
124
```

Impressão da árvore binária




```
----------------NULL
----------65
----------------NULL
-----60
----------------NULL
----------55
----------------NULL
50
----------------NULL
----------45
----------------NULL
-----40
----------------NULL
----------35
----------------NULL
```

Implementação de árvore binária de busca

```
125 int main(void) {
126     link Arvore = NULL; //inicializar a ABB vazia
127     int k;
128 
129     Arvore = inserir(Arvore, 50);
130     Arvore = inserir(Arvore, 40);
131     Arvore = inserir(Arvore, 60);
132     Arvore = inserir(Arvore, 35);
133     Arvore = inserir(Arvore, 45);
134     Arvore = inserir(Arvore, 55);
135     Arvore = inserir(Arvore, 65);
136 
137     imprimirPreOrdem(Arvore);
138     printf("\n\n");
139     imprimirInOrdem(Arvore, 0);
140 
141     printf("\n");
142     percursoPreOrdem(Arvore);
143     printf("\n");
144     percursoInOrdem(Arvore);
145     printf("\n");
146     percursoPosOrdem(Arvore);
147 
```

Implementação de árvore binária de busca

```
148     printf("\n\nInforme o valor que deseja procurar: ");
149     scanf("%d", &k);
150     if (buscar(Arvore, k) != NULL)
151        printf("O elemento %d foi encontrado na árvore.\n", k);
152     else
153        printf("O elemento %d não foi encontrado na árvore.\n", k);
154
155     printf("\nInforme o valor que deseja remover: ");
156     scanf("%d", &k);
157     Arvore = remover(Arvore, k);
158
159     imprimirInOrdem(Arvore, 0);
160
161     return 0;
162 }
```

Exercício

Implemente, em linguagem C, uma função para o percurso por nível (ou em largura) em uma árvore binária. (**Dica:** utilize uma fila)

Fim!